N89-15590

Graphic Simulation Test Bed for Robotics Applications in a Workstation Environment

J. Springfield, A. Mutammara, G. Karsai
G.E. Cook and J. Sztipanovits
Department of Electrical Engineering
Vanderbilt University, Nashville, TN 37235
K. Fernandez, Ph.D.
Automation and Robotics Research,
Marshall Space Flight Center, Huntsville, AL 35812

Abstract -- Graphical simulation is a cost-effective solution for developing and testing robots and their control systems. The availability of various high-performance workstations makes these systems feasible in everyday practice. Simulation offers preliminary testing of systems before their actual realizations, and it provides a framework for developing new control and planning algorithms. On the other hand, these simulation systems have to have the capability of incorporating various knowledge-based system components, e.g. task planners, representation formalisms, etc. They also should have an appropriate user interface, which makes possible the creation and control of simulation models.

ROBOSIM was developed jointly by MSFC and Vanderbilt University, first in a VAX environment. Recently, the system has been ported to an HP-9000 workstation equipped with an SRX graphics accelerator. The user interface of the system now contains a menu- and icon-based facility, as well as the original ROBOSIM language. The system is also coupled to a symbolic computing system based on Common Lisp, where knowledge-based functionalities are implemented. The knowledge-based layer uses various representation and reasoning facilities for programming and testing the control systems of robots.

Introduction

Robots are becoming increasingly important in every area of industry. Along with an increase in robot sophistication and payload capability there is an increase in the possibility of damage to the robot and to the workcell, especially during the development of the system and the algorithms or programs that will drive the robot. Simulation is even more important for robotic systems designed to operate in space. These robots, which are desgned to operate in zero-g, can not be tested in full on the ground. While robot simulation programs have existed for a while, the advent of high-speed graphics workstations allows real-time graphical simulation at a fraction of the cost as in the past.

ROBOSIM [1,2] was developed jointly between NASA and Vanderbilt University. ROBOSIM has been running on a DEC VAX 11/780 with the capability to use terminals with TEK4014 graphics compatibility. Interfaces for Evans & Sutherland PS3xx, GTI Poly 2000, and Silicon Grahphics IRIS are supported also.

ROBOSIM operates via an interpreted program that the user writes. The program consists of commands that create various solid and planar primitives that can be rotated and translated by other commands. In this way, the links of the robot are built up, with the relationships between the links following the Denavit-Hartenberg convention. ROBOSIM generates structures describing the physical structure, its kinematics, and its mass properties. With this information everything is known except for joint position, velocity, and torque limits, which are specified in the actual simulation. Using this information, all aspects of the simulation can be implemented, while reducing the possibility of data skewing, as the physical model, the kinematics, and the mass properties are calculated at the same time from the same program. All data is provided for collision detection, graphics display, and dynamics.

This full implementation of ROBOSIM has been ported to a Hewlett-Packard 350SRX graphics workstation. Several additional features have been added to exploit the capabilities of this workstation, such as the 3D graphics editor.

Graphics Editor

The capabilities of the workstation allow a much friendlier user-interface. X-Windows and the 3D graphics library (along with the special-purpose graphics hardware) provide the basis for a sophisticated means of designing robots. The editor utilizes menus and a mouse to provide simple methods to use the system. Also, a "box" of analog knobs allow the viewpoint to be changed, colors altered, and various other parameters concerning the graphic display to be set as desired. Although similar to some CAD systems, the editor is designed with ROBOSIM in mind.

The editor uses object oriented methods in its operation. Different types of objects can be created, such as boxes, cylinders, and custom designed objects. The reference frames of the links are also defined as objects. After creation, objects can be rotated, translated, deleted, and resized. Also, the objects can be attached together. This attachment is hierarchical in that there is a parent object and a child object. Any rotation or translation on a parent object propogates to any child object and its children as well. Once objects have been attached then a resizing operation on either will result in the relationship between them remaining constant. In this way complex, custom-designed, yet generic objects can be created. All that is necessary is to specify the dimensions of the components. This attachment allows more complex objects to be constructed without losing the ability to operate on the primitives.

Once the robot link has been built up, the structures can be saved for later editing. However, this editor will also generate a ROBOSIM program. The editor goes through the hierarchy, propogating every operation down. This yields a primitive object and a set of translations and rotations. Once this is achieved it is a fairly straigtforward procedure to automatically generate the ROBOSIM code. In this way the editor does not need to concern itself with mass properties or kinematics, as this is automatically produced by the ROBOSIM object compiler.

Object Compiler

The object compiler takes the output of the graphical editor and generates all of the data about a link: the polygonal model, the kinematic model, and the inertial data. Figure 1 shows a session during the editor in which a link is built. Figure 2 is a listing of the ROBOSIM code generated by the editor. And Figure 3 is an outline of the

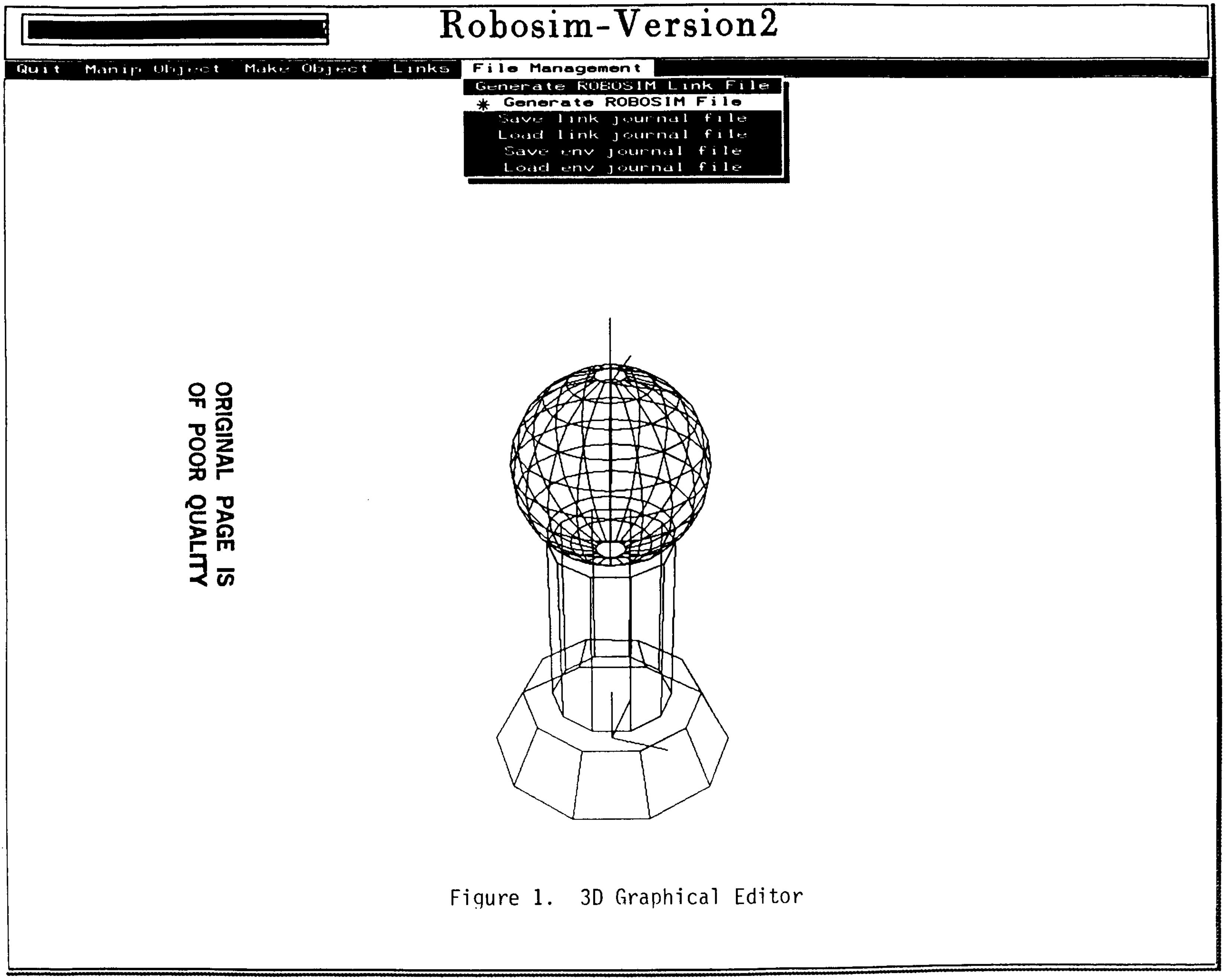

Figure 1. 3D Graphical Editor

```
LOOK-FROM X=-150., Y=50., Z=100.
LOOK-AT X=0., Y=0., Z=0.
CLEAR
STORE B
F-JOINT-I
      TRANSLATE X=0.000, Y=0.000, Z=-43.333
      ADD B
      STORE B
CLEAR
R-JOINT-I+1
      TRANSLATE X=0.000, Y=0.000, Z=23.811
      ADD B
      STORE B
CLEAR
SPHERE R=15.000
      TRANSLATE X=0.000, Y=0.000, Z=10.239
      ADD B
      STORE B
CLEAR
CYLINDER R=10.000, H=30.000
      TRANSLATE X=0.000, Y=0.000, Z=-18.333
      ADD B
      STORE B
CLEAR
TRUNCATED-CONE RL=20.000, RU=15.000, H=10.000
      TRANSLATE X=0.000, Y=0.000, Z=-40.000
      ADD B
      STORE B
CLEAR
LOAD B
STORE-LINK ROBOT.LOC
VIEW
END
```

Figure 2. ROBOSIM code generated by editor

Row	Col 1	Col 2	Col 3	Col 4
1	THETA	DZ	DA	ALPHA
2	JA1	JA2	JTYPE1	JTYPE2
3	AINERT (4X4)			
7	AJNT-I (4X4)			
11	AJNT-I+1 (4X4)			
15	AMAT (4X4)			
19	NVEC	UNUSED	UNUSED	UNUSED
20	X1	Y1	Z1	D1
21	X2	Y2	Z2	D2
:	:	:	:	:
NVEC+19	XNVEC	YNVEC	ZNVEC	DNVEC

Variable Definitions:

THETA	=	Denavit-Hartenberg parameter
DZ	=	Denavit-Hartenberg parameter
DA	=	Denavit-Hartenberg parameter
ALPHA	=	Denavit-Hartenberg parameter
JA1,JA2	=	joint defined flag
JTYPE-I,I+1	=	joint type -> Revolute,Prismatic,Fixed
AINERT	=	generalized link inertia
AJNT-I,I+1	=	transforms of input and output frames
AMAT	=	link's A-matrix
NVEC	=	number of vectors in list
Xi,Yi,Zi	=	x,y, and z component of vector
Di	=	move or draw vector

Figure 3. Data Structure of Link

data structure generated by ROBOSIM.

Simulator

The basis of the simulator is a library of C routines. These routines operate directly on the ROBOSIM structures. This allows the specifics to be hidden from the user, unless direct manipulation is required by the simulation. With these routines, one can implement fast, very specific simulations that are written in C and linked with the simulation library. However, one can also implement an interactive, general-purpose simulation in those cases where real-time simulation is not required.

These routines allow various robots, environments, and objects to be loaded separately and combined in any configuration. They allow commands similar to many robot programming languages to drive the robots. Collision detection can be turned on or off, and forces and torques can be returned to the simulation or used for control of the robot. Straight-line motion is included as well as a facility to move along any parametrically defined function. The joint angles can also be returned to the simulation for its use.

Although not included at this time, translators for many robot programming languages will be available in order to download developed programs directly to the physical robot.

Knowledge Representation Facilities

It is important to look at the ideas presented above, along with areas of future expansion, from the point of view of knowledge representation. Different choices in representation can result in changes in efficiency, flexibility, system requirements, and user-friendliness.

The techniques of Knowledge Representation (KR, for short) have been developed in the framework of Artificial Intelligence research. For engineering purposes KR techniques offer enhanced capabilities for modeling, where in addition to traditional numerical models, various symbolic models can also be used. The latter might include various declarative languages which represent technical concepts and entities [3].

This idea is applied in the robot simulation system as follows. Robot modelling and programming are supported by various declarative languages, which supply a knowledge-based description of (1) the robot, (2) its control system, (3) its environment, and (4) its task to be performed. Currently, these declarative languages are realized in a Lisp framework.

The geometric objects in robot modelling together constitute a graphic model of a robot or objects from its environment, and they can be used for various other considerations dependent on the objects, e.g. collision detection, dynamics, and task planning. For this application a declarative language has been defined, which supports the hierarchical representation of geometrical objects by using object--oriented programming techniques, as discussed previously.

The computations for the forward and inverse dynamics can be synthesized from the geometrical and physical model. This happens as follows: from the declarations describing the geometry and the physical properties a dataflow graph is synthesized, which represents the flow of computations needed to solve the dynamics problem. This graph is executed on the Multigraph Architecture (MA) [4,5], which makes possible the integration of symbolic and numeric computing in such a way, that the structure of a complex computation is represented in a declarative framework, while the computational primitives can be represented in terms of fast and efficient numerical algorithms.

Joint constraints and geometrical properties together are used for collision detection, where the detection algorithm also utilizes the representation schemes mentioned above.

The simulation of a robot system generates numerical and graphical output: numerical output contains information about joint angles, forces and torques, etc., while graphical output provides visual feedback for the designer. To control a robot which has been modelled using the facilities described above, one can create robot controller structures using a similar knowledge representation scheme. Here, again, a declarative language is used, which lets the designer describe a control system in terms of signal processing blocks, which implement various functionalities of the control system. This declarative language is also supported by a graphical editing technique, which uses icons to specify the blocks, and they can be connected to form a signal processing graph. The resulting declarations are interpreted to generate the running signal processing system using the Multigraph Architecture.

The tasks to be performed are represented in declarative terms also. Using hierarchical decomposition, robot control sequences are represented in symbolic form, from which a task planner synthesizes the actual sequence of control actions needed to accomplish a certain goal. The hierarchical planner generates a chain of objects, which represent the steps to be performed. Each such step is linked to a low-level control scheme which implements that step. Now if the step has successfully been executed (i.e. the low-level control scheme has not signalled an error) the execution proceeds, while if there were a problem, the execution (1) might go along either a new path, or (2) a completely new sequence is synthesized together with its low-level control blocks. This dynamic replanning might influence the computational model of control, dynamically changing the structure of the control system. One example might be as follows: if the collision detection scheme signals an error to the controller system, that event might initiate the restarting of the task planner to re-generate or structurally modify the step sequence and/or its associated controller. This flexible way of representing tasks, task steps and low-level controllers was made possible by using the integrated computational model of the MA.

Conclusions

The need for competent, comprehensive robot simulators has been well established. In order to provide adequate simulation capability, the system must allow graphical, as well as dynamic, simulation. The new generation of graphics workstations provide for extremely fast graphics output, while freeing the main processor for kinematic and dynamic calculations. However, mere number crunching is not enough to create a better simulator. Symbolic planners and other such AI programs are necessary in order to fully meet the demanding requirements that are being asked for.

References

1. Fernandez, K.R. and Cook, G.E., "Use of Computer Graphic Simulation Techniques for Robot Control System Development", IEEE Computer Society, Proc. 18th Southestern Symposium on System Theory, Knoxville, TN, April 7-8, 1986, pp.433-441.

2. Fernandez, K.R., "The Use of Computer Graphic Simulation in the Development of Robotic Systems", Acta Astronautica, Vol. 17, No.1, pp.115-122, 1988.

3. Karsai, G., "Declarative Programming Techniques for Engineering Problems", Ph.D. Thesis, Vanderbilt University, August 1988.

4. Sztipanovits, J., "Execution Environment for Intelligent Real-time Control Systems," Proc. of the NASA/JPL Symposium on Telerobotics, 24 pgs., Pasadena, CA, 1987.

5. Biegl, C.A., "Design and Implementation of an Execution Environment for Knowledge-Based Systems", Ph.D. Thesis, Vanderbilt University, December 1988.